Formularium Probleemoplossend Denken II

Stijn Lievens
Hogeschool Gent
2018–2019

Kansen

Eigenschap (1.30, Regel van Bayes) Gegeven een gebeurtenis A met n elkaar wederzijds uitsluitende oorzaken B_i, i.e. $B_i \cap B_j = \emptyset$ als $i \neq j$ en $\Omega = \cup_{i=1}^{n} B_i$, dan geldt voor elke j dat

$$\mathbb{P}(B_j \mid A) = \frac{\mathbb{P}(B_j)\,\mathbb{P}(A \mid B_j)}{\mathbb{P}(A)} = \frac{\mathbb{P}(B_j)\,\mathbb{P}(A \mid B_j)}{\sum_{i=1}^{n} \mathbb{P}(B_i)\,\mathbb{P}(A \mid B_i)}.$$ ∎

Toevalsveranderlijken

Definitie (2.22) De VERWACHTINGSWAARDE van een discrete toevalsveranderlijke X wordt genoteerd als μ_X of $\mathrm{E}(X)$, en is een gewogen gemiddelde van de waarden x_i die X kan aannemen met de respectievelijke kansen als gewichten. In formulevorm:

$$\mathrm{E}(X) = \sum_i x_i\,\mathbb{P}(X = x_i) = \sum_i x_i f_X(x_i).$$ ∎

Definitie (2.24) De VARIANTIE van een discrete toevalsveranderlijke X, genoteerd σ_X^2, is de gewogen gemiddelde kwadratische afwijking t.o.v. zijn verwachtingswaarde. In symbolen:

$$\sigma_X^2 = \sum_i (x_i - \mu_X)^2\,\mathbb{P}(X = x_i) = \sum_i (x_i - \mu_X)^2 f_X(x_i).$$ ∎

Opmerking (2.25) De STANDAARDAFWIJKING, genoteerd σ_X, is de vierkantswortel van de variantie. ∎

Definitie (2.28) De verwachtingswaarde $\mathrm{E}(X)$ (of μ_X) van een continue toevalsveranderlijke X wordt gedefinieerd als:

$$\mu_X = \int_{-\infty}^{+\infty} x f_X(x)\,\mathrm{d}x,$$

terwijl de variantie σ_X^2 wordt gegeven door

$$\sigma_X^2 = \int_{-\infty}^{+\infty} (x-\mu_X)^2 f_X(x)\,\mathrm{d}x.$$ ∎

Kansverdelingen

Eigenschap (3.3, Verwachtingswaarde en variantie) Als X een toevalsveranderlijke is die een Bernoulliverdeling volgt met kans op succes p, dan geldt

$$\mu_X = p \quad \text{en} \quad \sigma_X^2 = p(1-p).$$ ∎

Eigenschap (3.5, Kansfunctie) Veronderstel dat $X \sim B(n,p)$, dan wordt de kansfunctie van X gegeven door:

$$f_X(k) = \mathbb{P}(X=k) = C_n^k p^k q^{n-k} = \binom{n}{k} p^k q^{n-k}, \quad \text{als } 0 \leq k \leq n.$$ ∎

Eigenschap (3.7, Verwachtingswaarde en variantie) Als $X \sim B(n,p)$ dan geldt

$$\mu_X = np \quad \text{en} \quad \sigma_X^2 = np(1-p).$$ ∎

Eigenschap (3.10, Kansfunctie) Als X geometrisch verdeeld is met parameter p, dan wordt de kansfunctie f_X gegeven door

$$f_X(k) = \mathbb{P}(X=k) = q^{k-1}p, \quad \text{voor } k \geq 1.$$ ∎

Eigenschap (3.12, Verwachtingswaarde en variantie) Als X geometrisch verdeeld is met parameter p dan geldt:

$$\mu_X = \frac{1}{p} \quad \text{en} \quad \sigma_X^2 = \frac{q}{p^2}.$$ ∎

Eigenschap (3.16, Kansfunctie) Als X Poisson-verdeeld is met parameter λ, dan wordt zijn kansfunctie gegeven door

$$f_X(k) = \mathbb{P}(X = k) = e^{-\lambda}\frac{\lambda^k}{k!}, \quad \text{met} \quad k \geq 0.$$

Eigenschap (3.18, Verwachtingswaarde en variantie) Als de toevalsveranderlijke X Poisson-verdeeld is met parameter λ, dan geldt

$$\mu_X = \lambda \quad \text{en} \quad \sigma_X^2 = \lambda.$$

Eigenschap (3.20, Kansdichtheid) Als X een continue uniforme verdeling volgt met grenzen a en b, dan wordt de kansdichtheid van X gegeven door

$$f_X(x) = \begin{cases} \dfrac{1}{b-a} & \text{als } a \leq x \leq b, \\ 0 & \text{anders.} \end{cases}$$

Eigenschap (3.21, Verwachtingswaarde en variantie) Als X een continue uniforme verdeling volgt met grenzen a en b dan geldt

$$\mu_X = \frac{(a+b)}{2} \quad \text{en} \quad \sigma_X^2 = \frac{(b-a)^2}{12}.$$

Eigenschap (3.23, Kansdichtheid en kansverdelingsfunctie) De kansdichtheid en kansverdelingsfunctie van een exponentieel verdeelde veranderlijke T met parameter λ worden respectievelijk gegeven door:

$$f_T(t) = \begin{cases} 0 & \text{als } t < 0 \\ \lambda e^{-\lambda t} & \text{als } t \geq 0, \end{cases}$$

en

$$F_T(t) = \begin{cases} 0 & \text{als } t < 0 \\ 1 - e^{-\lambda t} & \text{als } t \geq 0. \end{cases}$$

Eigenschap (3.24, Verwachtingswaarde en variantie) Als T een toevalsveranderlijke is die exponentieel verdeeld is met parameter λ dan geldt

$$\mu_T = \frac{1}{\lambda} \quad \text{en} \quad \sigma_T^2 = \frac{1}{\lambda^2}.$$

Graafalgoritmes

Generiek zoeken

Algoritme 5.1 Generiek zoeken in een graaf.

Invoer Een gerichte of ongerichte graaf $G = (V, E)$ met orde $n > 0$. Een knoop s waarvan het zoeken vertrekt. De knopen zijn genummerd van 1 tot n, i.e. $V = \{1, 2, \ldots, n\}$.

Uitvoer Een array D met $D[v] = \textbf{true}$ als en slechts als er een pad bestaat van s naar v.

```
1: function ZoekGeneriek(G, s)
2:     D ← [false, false, ..., false]                ▷ n keer false
3:     D[s] ← true                                   ▷ markeer s
4:     while ∃(u, v): D[u] = true ∧ D[v] = false do
5:         kies een boog (u, v) met D[u] = true ∧ D[v] = false
6:         D[v] ← true                               ▷ markeer v
7:     end while
8:     return D
9: end function
```

Breedte-Eerst zoeken

Algoritme 5.2 Breedte-eerst zoeken in een graaf.

Invoer Een gerichte of ongerichte graaf $G = (V, E)$ met orde $n > 0$. Een knoop s waarvan het zoeken vertrekt. De knopen zijn genummerd van 1 tot n, i.e. $V = \{1, 2, \ldots, n\}$.

Uitvoer Een array D met $D[v] =$ **true** als en slechts als er een pad bestaat van s naar v.

```
1:  function BreedteEerst(G, s)
2:      D ← [false, false, ..., false]                ▷ n keer false
3:      D[s] ← true                                    ▷ markeer s
4:      Q.init()                                       ▷ wachtrij van knopen
5:      Q.enqueue(s)
6:      while Q ≠ ∅ do
7:          v ← Q.dequeue()
8:          for all w ∈ buren(v) do
9:              if D[w] = false then                   ▷ w nog niet ontdekt
10:                 D[w] ← true
11:                 Q.enqueue(w)
12:             end if
13:         end for
14:     end while
15:     return D
16: end function
```

Diepte-Eerst Zoeken

Algoritme 5.3 Diepte-eerst zoeken in een graaf.

Invoer Een gerichte of ongerichte graaf $G = (V, E)$ met orde $n > 0$. Een knoop s waarvan het zoeken vertrekt. De knopen zijn genummerd van 1 tot n, i.e. $V = \{1, 2, \ldots, n\}$.

Uitvoer Een array D met $D[v] =$ **true** als en slechts als er een pad bestaat van s naar v.

```
1:  function DiepteEerst(G, s)
2:      D ← [false, false, ..., false]                ▷ n keer false
3:      DiepteEerstRecursief(G, s, D)
4:      return D
5:  end function
6:  function DiepteEerstRecursief(G, v, D)
7:      D[v] ← true                                   ▷ markeer v
8:      for all w ∈ buren(v) do
9:          if D[w] = false then                      ▷ w nog niet ontdekt
10:             DiepteEersteRecursief(G, w, D)
11:         end if
12:     end for
13: end function
```

Topologisch Sorteren

Algoritme 5.4 Topologisch sorteren van een graaf.

Invoer Een gerichte graaf $G = (V, E)$ met orde $n > 0$. De knopen zijn genummerd van 1 tot n, i.e. $V = \{1, 2, \ldots, n\}$.

Uitvoer Een topologische sortering van G indien mogelijk, **false** anders.

```
1:  function SorteerTopologisch(G)
2:      global cycleDetected ← false                    ▷ globale variabele
3:      D ← [0, 0, ..., 0]                              ▷ n keer 0
4:      S ← ∅                                           ▷ S is lege lijst
5:      for all s ∈ V do
6:          if D[s] = 0 then                            ▷ s nog niet gezien
7:              DfsTopo(G, s, D, S)                     ▷ S en D referentieparameters
8:              if cycleDetected = true then            ▷ controleer op cykel
9:                  return false
10:             end if
11:         end if
12:     end for
13:     return S
14: end function
15: function DfsTopo(G, v, D, S)
16:     D[v] ← 1                                        ▷ markeer v als 'bezig'
17:     for all w ∈ buren(v) do
18:         if D[w] = 0 ∧ cycleDetected = false then    ▷ w nog niet ontdekt
19:             DfsTopo(G, w, D, S)
20:         else if D[w] = 1 then                       ▷ cykel ontdekt w ⇝ v → w
21:             cycleDetected ← true
22:         end if
23:     end for
24:     D[v] ← 2                                        ▷ markeer v als 'voltooid'
25:     voeg v vooraan toe aan S                        ▷ ken rangnummer toe aan v
26: end function
```

Kortste Pad Algoritmen

Kortste Pad in een Ongewogen Graaf

Algoritme 5.5 Kortste pad in een ongewogen graaf

Invoer Een gerichte of ongerichte ongewogen graaf $G = (V, E)$ met orde $n > 0$. Een knoop s waarvan het zoeken vertrekt. De knopen zijn genummerd van 1 tot n, i.e. $V = \{1, 2, \ldots, n\}$.

Uitvoer De array D met $D[v]$ de kortste afstand van s tot v; als $D[v] = \infty$ dan is er geen pad van s naar v.

```
1:  function KortstePadOngewogen(G, s)
2:      D ← [∞, ∞, ..., ∞]                                  ▷ n keer ∞
3:      D[s] ← 0                  ▷ kortste pad van s naar zichzelf heeft lengte 0
4:      Q.init()                                             ▷ wachtrij van knopen
5:      Q.enqueue(s)
6:      while Q ≠ ∅ do
7:          v ← Q.dequeue()
8:          for all w ∈ buren(v) do
9:              if D[w] = ∞ then                             ▷ w nog niet ontdekt
10:                 D[w] ← D[v] + 1
11:                 Q.enqueue(w)
12:             end if
13:         end for
14:     end while
15:     return D
16: end function
```

Dijkstra's Algoritme

Algoritme 5.6 Het algoritme van Dijkstra

Invoer Een gerichte of ongerichte gewogen graaf $G = (V, E)$ met orde $n > 0$. Alle gewichten zijn positief. Een knoop s waarvan het zoeken vertrekt. De knopen zijn genummerd van 1 tot n, i.e. $V = \{1, 2, \ldots, n\}$.

Uitvoer De array D met $D[v]$ de kortste afstand van s tot v; als $D[v] = \infty$ dan is er geen pad van s naar v.

```
1:  function DIJKSTRA(G, s)
2:      D ← [∞, ∞, ..., ∞]                                   ▷ n keer ∞
3:      D[s] ← 0                          ▷ kortste pad van s naar zichzelf heeft lengte 0
4:      Q ← V                  ▷ knopen waarvan kortste afstand nog niet is bepaald
5:      while Q ≠ ∅ do
6:          zoek v ∈ Q waarvoor D[v] minimaal is (voor knopen in Q)
7:          verwijder v uit Q
8:          for all w ∈ buren(v) ∩ Q do
9:              if D[w] > D[v] + gewicht(v, w) then
10:                 D[w] ← D[v] + gewicht(v, w)                ▷ korter pad naar w
11:             end if
12:         end for
13:     end while
14:     return D
15: end function
```

Minimale Kost Opspannende Bomen

Prims Algoritme

Algoritme 5.7 Prims algoritme voor een minimale kost opspannende boom

Invoer Een ongerichte gewogen graaf $G = (V, E)$ met orde $n > 0$. De knopen zijn genummerd van 1 tot n, i.e. $V = \{1, 2, \ldots, n\}$.

Uitvoer Een verzameling T van bogen die een minimale kost opspannende boom is.

```
1:  function PRIM(G)
2:      D ← [false, false, ..., false]                    ▷ n keer false
3:      D[1] ← true                                       ▷ kies knoop 1 als startknoop
4:      T ← ∅                                             ▷ gekozen bogen
5:      while ∃(u, v): D[u] = true ∧ D[v] = false do
6:          kies boog (u, v) met D[u] = true ∧ D[v] = false met minimaal gewicht
7:          D[v] ← true
8:          T ← T ∪ {(u, v)}                              ▷ Voeg boog (u, v) toe aan boom
9:      end while
10:     return T
11: end function
```

Kruskals Algoritme

Algoritme 5.8 Kruskals algoritme voor een minimale kost opspannende boom

Invoer Een ongerichte gewogen graaf $G = (V, E)$ met grootte $n > 0$. De knopen zijn genummerd van 1 tot n, i.e. $V = \{1, 2, \ldots, n\}$.

Uitvoer Een verzameling T van bogen die een minimale kost opspannende boom is.

```
1:  function KRUSKAL(G)
2:      T ← ∅                                   ▷ Start met lege boom
3:      E' ← sorteer E volgens stijgend gewicht
4:      for all e' ∈ E' do
5:          if T ∪ e' heeft geen cykel then
6:              T ← T ∪ e'
7:          end if
8:      end for
9:      return T
10: end function
```

Branch and bound methode voor ILP problemen

We bekijken enkel maximalisatieproblemen.

1. Stel $D^* = -\infty$ (nog geen oplossing gevonden).
2. Los de LP-relaxatie van het oorspronkelijke probleem op en vind op die manier een bovengrens voor de D-waarde van het oorspronkelijke ILP-probleem. Voeg dit probleem toe aan de lijst van open problemen (als de LP-relaxatie een oplossing heeft).
3. Herhaal nu de volgende drie stappen todat de oplossing werd bekomen of totdat er geen open problemen meer zijn.
   (a) Kies een probleem van de open lijst, bv. het probleem met de beste (grootste) bovengrens. Splits (**branch**) het op in (twee) deelproblemen door bv. de eerste variabele x_j met waarde x_j^* te beschouwen die niet aan de geheeltalligheidseisen voldoet en voeg de beperkingen $x_j \leq \lfloor x_j^* \rfloor$ en $x_j \geq \lceil x_j^* \rceil$ toe aan het probleem dat van de open lijst werd gehaald.

(b) Bereken een bovengrens (**bound**) voor deze problemen (bv. met de simplex-methode).

(c) Doorloop de volgende stappen voor deze twee problemen:

i. De bovengrens voor het probleem is kleiner of gelijk aan D^*. Dit probleem mag men negeren want hier kunnen geen betere oplossingen gevonden worden.

ii. Het probleem heeft geen aanvaardbare oplossingen; het probleem mag genegeerd worden.

iii. De oplossing van het probleem voldoet aan alle geheeltalligheidseisen. Indien de D-waarde beter is dan D^* dan wordt D^* aangepast (en onthouden we deze oplossing). In dit laatste geval worden alle problemen op de open lijst met een bovengrens kleiner of gelijk aan D^* van de open lijst verwijderd.

iv. Indien geen enkel van de voorgaande drie voorwaarden voldaan is, dan wordt het probleem aan de open lijst toegevoegd.

Branch and bound voor het knapzakprobleem

De branch and bound methode kan ook toegepast worden voor knapzakproblemen. Merk op:

1. Bij een knapzakprobleem kan men echter starten met de oplossing gevonden via het *benaderend gulzig* algoritme waarbij de items geordend werden in dalende volgorde van waarde-per-gewicht ratio.

2. Om de bovengrens voor de deelproblemen te vinden hoeft men het simplexalgoritme niet uit te voeren. Het gulzig algoritme waarbij ook fractionele items zijn toegelaten vindt steeds de optimale oplossing.